ECE 545
Lecture 12

Design of Controllers Using ASM Charts

George Mason University

1

Recommended reading

- P. Chu, *RTL Hardware Design using VHDL*

Chapter 11, Register Transfer Methodology: Principle

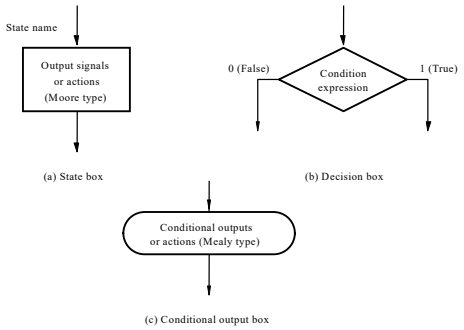
Chapter 12, Register Transfer Methodology: Practice

2

Algorithmic State Machine Charts

3

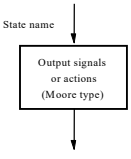
Elements used in ASM charts (1)



4

State Box

- **State box** – represents a state.
- Equivalent to a node in a state diagram or a row in a state table.
- Contains register transfer actions or output signals
- **Moore-type outputs are listed inside of the box.**
- It is customary to write only the name of the signal that has to be asserted in the given state, e.g., z instead of z<=1.
- Also, it might be useful to write an action to be taken, e.g., count <= count + 1, and only later translate it to asserting a control signal that causes a given action to take place (e.g., enable signal of a counter).

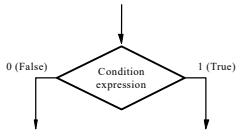


5

Decision Box

- **Decision box** – indicates that a given condition is to be tested and the exit path is to be chosen accordingly.

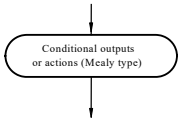
  The condition expression may include one or more inputs to the FSM.



6

Conditional Output Box

- **Conditional output box**
- **Denotes output signals that are of the Mealy type.**
- The condition that determines whether such outputs are generated is specified in the decision box.



7

ASM Block

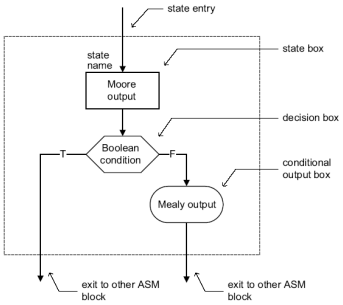


Figure 10.4 ASM block.

- One state box
- One or more (optional) decision boxes: with T (1) or F (0) exit paths
- One or more (optional) conditional output boxes: for Mealy outputs

8

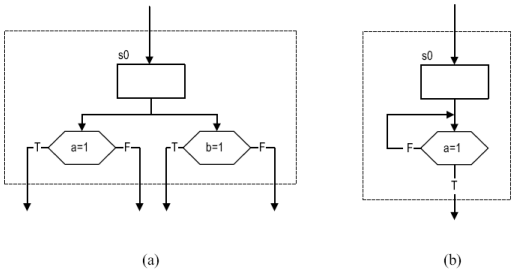
ASM Chart Rules

- Difference between a regular flowchart and an ASM chart:
  - Transition governed by clock
  - Transition occurs between ASM blocks
- Basic rules:
  - For a given input combination, there is one unique exit path from the current ASM block
  - Any closed loop in an ASM chart must include a state box

Based on RTL Hardware Design by P. Chu

9

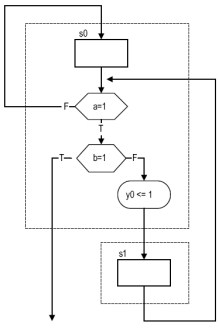
Incorrect ASM Charts



Based on RTL Hardware Design by P. Chu

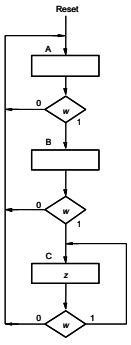
10

Correct ASM Chart



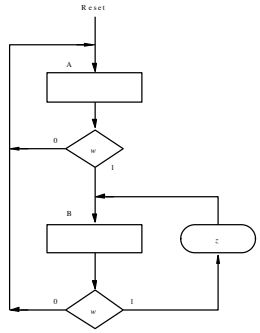
11

Example 1: ASM Chart



12

Example 2: ASM Chart



13

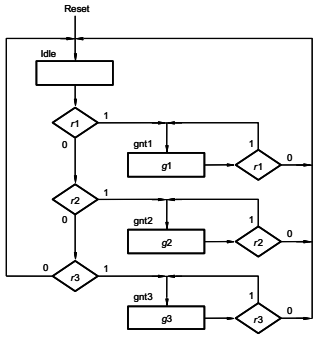
13

Exercise

Assuming ASM chart given on the next slide, supplement timing waveforms given in the answer sheet with the correct values of signals **State, g1, g2, g3**, in the interval from 0 to 575 ns. Assume zero delays.

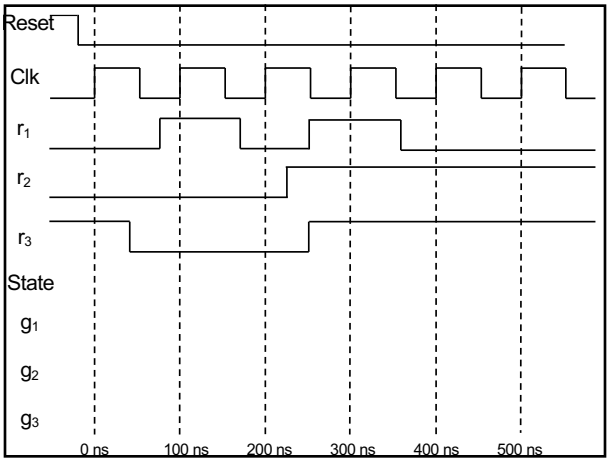
14

ASM Chart



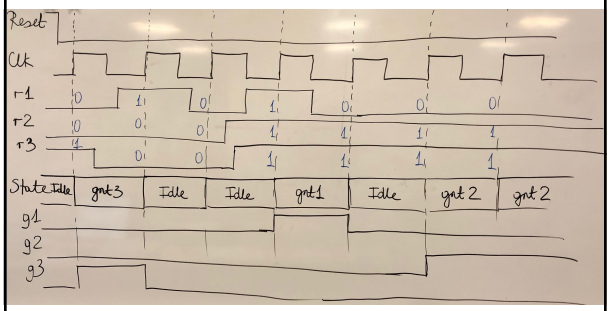
15

15



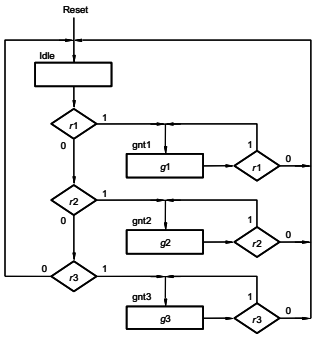
16

Solution



17

ASM Chart

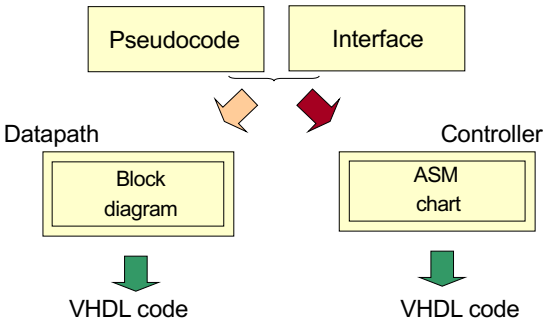


18

Design of Controllers

19

Hardware Design with RTL VHDL



20

Steps of the Design Process Introduced in Class Today

1. Text description
2. Interface
3. Pseudocode
4. Block diagram of the Datapath
5. Interface with the division into Datapath and Controller
6. **ASM chart of the Controller**
7. **RTL VHDL code of** the Datapath, **Controller**, and Top-Level Unit
8. **Testbench of** the Datapath, **Controller**, and Top-Level Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing

21

Class Exercise 1
STATISTICS

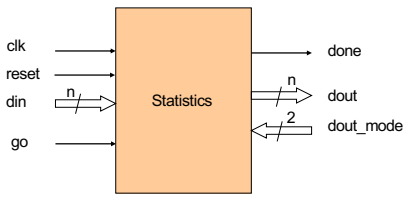
22

Pseudocode

```
no_1 = no_2 = no_3 = sum = 0
for i=0 to k-1 do
    sum = sum + din
    if  din > no_1 then
            no_3 = no_2
            no_2 = no_1
            no_1 = din
    elseif (din > no_2) then
            no_3 = no_2
            no_2 = din
    elseif (din > no_3) then
            no_3 = din
    end if
end for
avr = sum / k
```

23

Circuit Interface

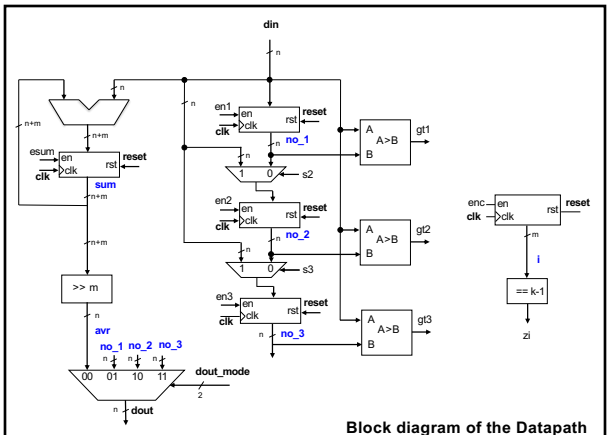


24

Interface Table

Port	Width	Meaning
clk	1	System clock.
reset	1	System reset.
din	n	Input Data.
go	1	Control signal indicating that the first input is ready. Active for one clock cycle.
done	1	Signal set to high after the output is ready.
dout	n	Output dependent on the dout_mode input.
dout_mode	2	Control signal determining value available at the output. 00: avr, 01: no_1, 10: no_2, 11: no_3.

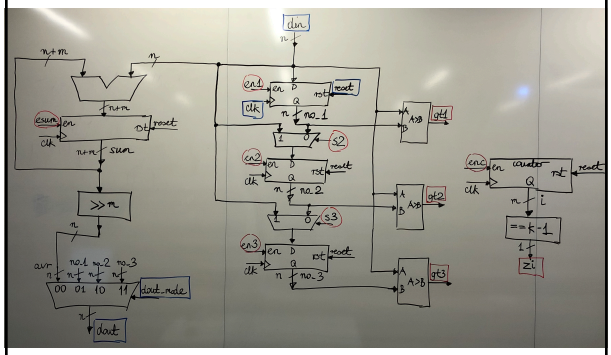
25



Block diagram of the Datapath

26

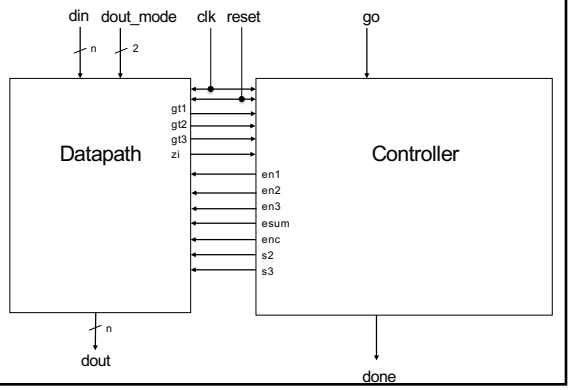
Block Diagram



27

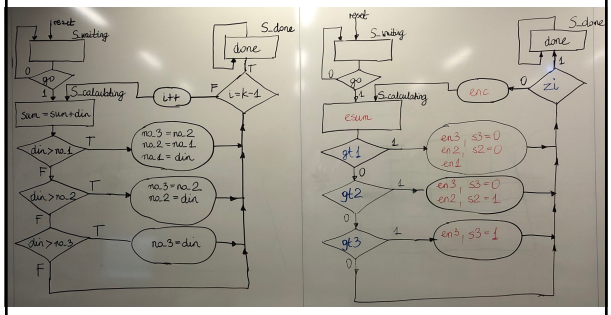
27

Interface with the division into the Datapath and Controller



28

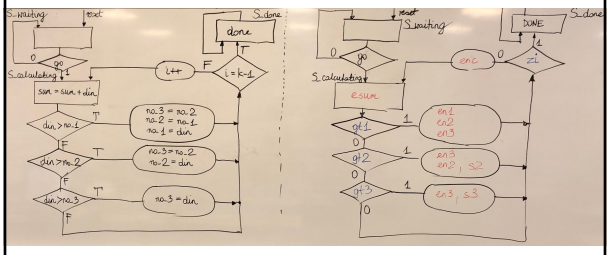
ASM Charts



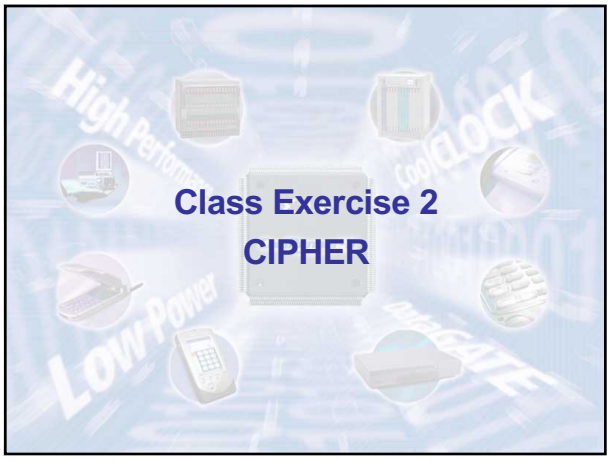
29

Simplified ASM Charts

The default value of select signals assumed to be 0



30



31

Pseudocode

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
      for i = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            U = (D*(2D + 1)) <<< k
            A = ((A ⊕ T) <<< U) + S[2i]
            C = ((C ⊕ U) <<< T) + S[2i + 1]
            (A, B, C, D)  =  (B, C, D, A)
      }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

32

32

Notation

w: word size, e.g., w=32 (constant)
k: $\log_2(w)$ (constant)
A, B, C, D, U, T: w-bit variables
I3, I2, I1, I0: Four w-bit words of the input I
r: number of rounds (constant)
O: output of the size of 4w bits
S[j] : 2r+4 round keys stored in two RAMs.
Each key is a w-bit word.
The first RAM stores values of S[j=2i], i.e., only round keys with even indices. The second memory stores values of S[j=2i+1], i.e., only round keys with odd indices.

33

33

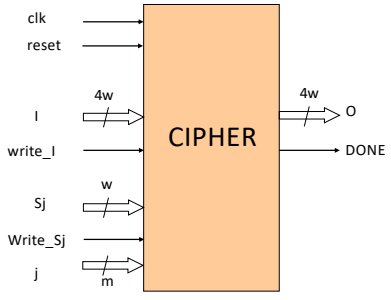
Operations

⊕ : XOR
+ : addition modulo 2^w
– : subtraction modulo 2^w
*** :** multiplication modulo 2^w
X <<< Y : rotation of X to the left by the number of positions given in Y
X >>> Y : rotation of X to the right by the number of positions given in Y
(A, B, C, D) : Concatenation of A, B, C, and D.

34

34

Circuit Interface



35

35

Interface Table

Port	Width	Meaning
clk	1	System clock.
reset	1	System reset – clears internal registers.
I	4w	Input block.
write_I	1	Synchronous write control signal for the input block I. After the block I is written to the CIPHER unit, the encryption of I starts automatically.
Sj	w	Round key S[j] loaded to one of the two internal memories.
write_Sj	1	Synchronous write control signal for the round key S[j].
j	m	Index of the round key S[j] loaded using input Sj.
O	4w	Output block.
DONE	1	Asserted for one clock cycle when the output O is ready.

Note:
m is a size of index j. It is a minimum integer, such that $2^m - 1 \geq 2r+3$.

36

36

Protocol (1)

An external circuit first loads all round keys
S[0], S[1], S[2], …, S[2r+2], [2r+3]
to the two internal memories of the CIPHER unit.

The first memory stores values of S[j=2i], i.e., only round keys with even indices. The second memory stores values of S[j=2i+1], i.e. only round keys with odd indices.

Loading round keys is performed using inputs: Sj, j, write_Sj, clk.

Then, the external circuits, loads an input block I to the CIPHER unit, using inputs: I, write_I, clk.

After the input block I is loaded to the CIPHER unit, the encryption starts automatically.

37

37

Protocol (2)

When the encryption is completed, signal DONE becomes active, and the output O changes to the new value of the ciphertext.

The output O keeps the last value of the ciphertext at the output, until the next encryption is completed. Before the first encryption is completed, this output should be equal to zero.

38

38

Assumptions

- 2r+4 clock cycles are used to load round keys to internal RAMs
- one round of the main for loop of the pseudocode executes in one clock cycle
- you can access only one position of each internal memory of round keys per clock cycle

As a result, the encryption of a single input block I should last r+2 clock cycles.

39

39

Pseudocode

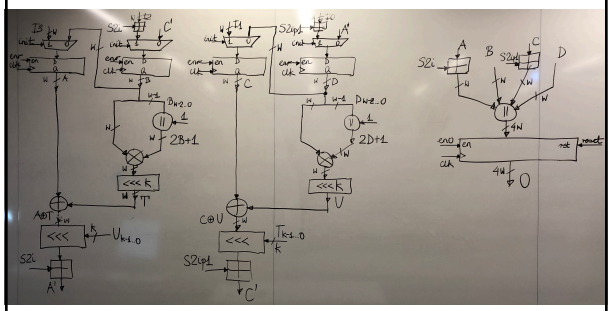
Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
      for i = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            U = (D*(2D + 1)) <<< k
            A' = ((A ⊕ T) <<< U) + S[2i]
            C' = ((C ⊕ U) <<< T) + S[2i + 1]
            (A, B, C, D)  =  (B, C', D, A')
      }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

40

40

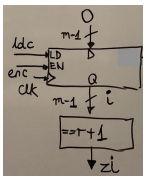
Datapath – Initialization, Main Loop, & PostProcessing



41

41

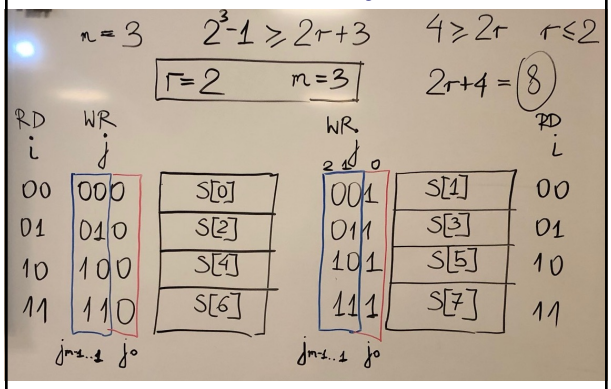
Datapath – Loop Counter



42

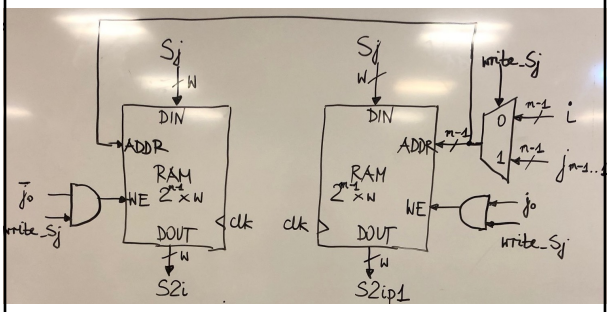
42

Memories of Round Keys – Contents



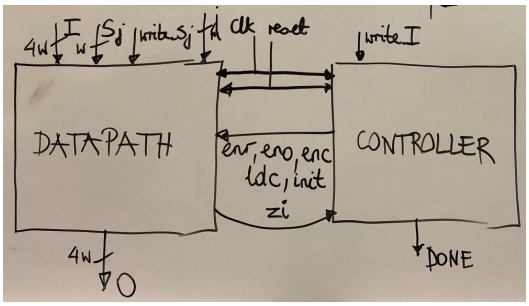
43

Datapath – Memories of Round Keys



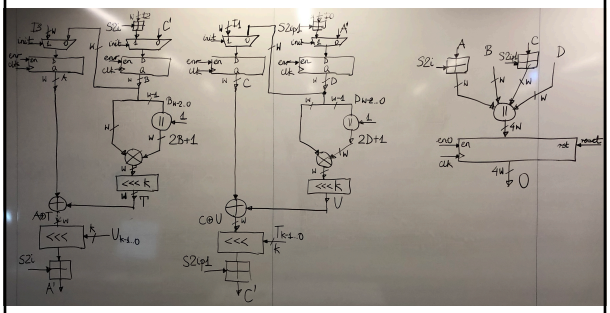
44

Interface with the Division into the Datapath and Controller



45

Datapath – Initialization, Main Loop, & PostProcessing



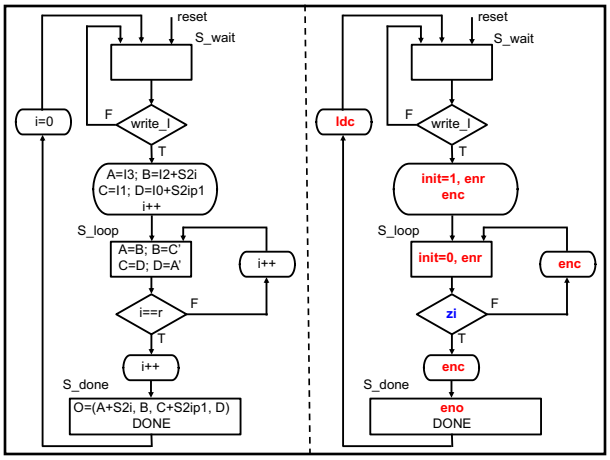
46

Pseudocode

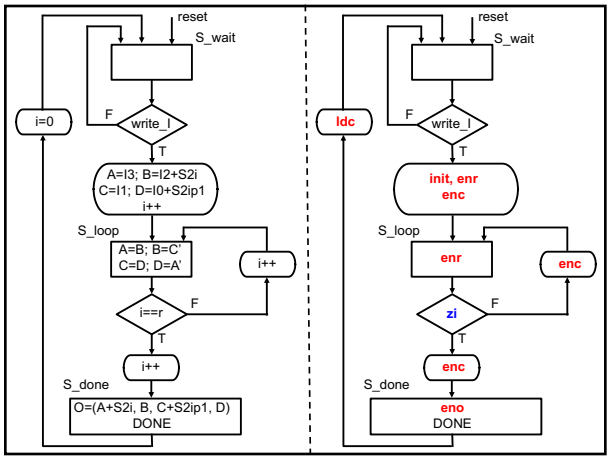
Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
      for i = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            U = (D*(2D + 1)) <<< k
            A' = ((A ⊕ T) <<< U) + S[2i]
            C' = ((C ⊕ U) <<< T) + S[2i + 1]
            (A, B, C, D)  =  (B, C', D, A')
      }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```


47

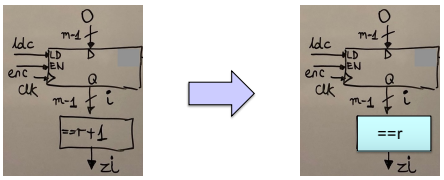


48



49

Datapath – Loop Counter



50

50

Timing Analysis – Key Setup & Encryption

Notation:
T_{CLK} – clock period in ns
N_M – number of encrypted message blocks I

$T_{KeySetup}[cycles] = 2r+4$

$T_{KeySetup}[ns] = (2r+4) \cdot T_{CLK}$

$T_{Encryption}(N_M)[cycles] = (r+2) \cdot N_M$

$T_{Encryption}(N_M)[ns] = (r+2) \cdot N_M \cdot T_{CLK}$

51

51

Timing Analysis - Throughput

Notation:
T_{CLK} – clock period in ns
N_M – number of encrypted message blocks I
w – word size

$$Thr_{Encryption}(N_M)[Gbit/s] = \frac{4 \cdot w \cdot N_M}{(r+2) \cdot N_M \cdot T_{CLK}} = \frac{4 \cdot w}{(r+2) \cdot T_{CLK}}$$

r & w constants, determined by the security analysis of the cipher
For the security equivalent to AES-128, r=20 & w=32

T_{CLK} dependent on w, returned by the FPGA tools based on the static timing analysis of the critical path

52

52



53

Pseudocode

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
      for i = 1 to r do
      {
            T = (B*(2B + 1)) <<< k
            U = (D*(2D + 1)) <<< k
            A = ((A ⊕ T) <<< U) + S[2i]
            C = ((C ⊕ U) <<< T) + S[2i + 1]
            (A, B, C, D)  =  (B, C, D, A)
      }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

54

54

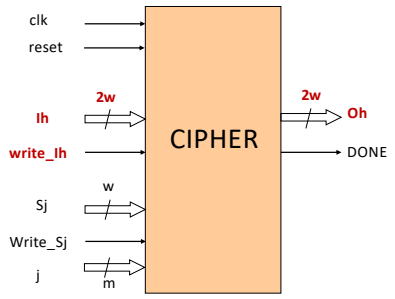
Notation

w: word size, e.g., w=32 (constant)
k: $\log_2(w)$ (constant)
A, B, C, D, U, T: w-bit variables
I3, I2, I1, I0: Four w-bit words of the input I
r: number of rounds (constant)
O: output of the size of 4w bits
S[j] : 2r+4 round keys stored in **one RAM**.
Each key is a w-bit word.

55

55

Circuit Interface



56

56

Interface Table

Port	Width	Meaning
clk	1	System clock.
reset	1	System reset – clears internal registers.
Ih	2w	One half of an Input block. I3:I2 first, I1:I0 second.
write_Ih	1	Synchronous write control signal for a half of the input block Ih. After two halves of the input block I are written to the CIPHER unit, the encryption of I starts automatically.
Sj	w	Round key S[j] loaded to one of the two internal memories.
write_Sj	1	Synchronous write control signal for the round key S[j].
j	m	Index of the round key S[j] loaded using input Sj.
Oh	2w	One half of an Output block. O3:O2 first, O1:O0 second.
DONE	1	Asserted for one clock cycle when the output O is ready.

Note:
m is a size of index j. It is a minimum integer, such that $2^m - 1 \geq 2r+3$.

57

57

Protocol (1)

An external circuit first loads all round keys
S[0], S[1], S[2], …, S[2r+2], [2r+3]
to **the internal memory** of the CIPHER unit.

Loading round keys is performed using inputs: Sj, j, write_Sj, clk.

Then, the external circuits, loads an input block I to the CIPHER unit, **in two clock cycles**, using inputs: **Ih**, **write_Ih**, clk.

After the input block I is loaded to the CIPHER unit, the encryption starts automatically.

58

58

Protocol (2)

When the encryption is completed, signal DONE becomes active, and the output O changes to the new value of the ciphertext.

The output O keeps the last value of the ciphertext at the output, until the next encryption is completed. Before the first encryption is completed, this output should be equal to zero.

59

59

Assumptions

- 2r+4 clock cycles are used to load round keys to internal RAMs
- one round of the main for loop of the pseudocode executes in **4 clock cycles**
- you can access only one position of **the internal memory** of round keys per clock cycle

As a result, the encryption of a single input block I should last **4r+4** clock cycles.

60

60

Pseudocode

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[0]
D = D + S[1]
    for i = 1 to r do
    {
        T = (B*(2B + 1)) <<< k
        U = (D*(2D + 1)) <<< k
        A = ((A ⊕ T) <<< U) + S[2i]
        C = ((C ⊕ U) <<< T) + S[2i + 1]
        (A, B, C, D)  =  (B, C, D, A)
    }
A = A + S[2r + 2]
C = C + S[2r + 3]
O = (A, B, C, D)
```

61

61

Pseudocode

i represents an index of a round key during encryption, not a round number

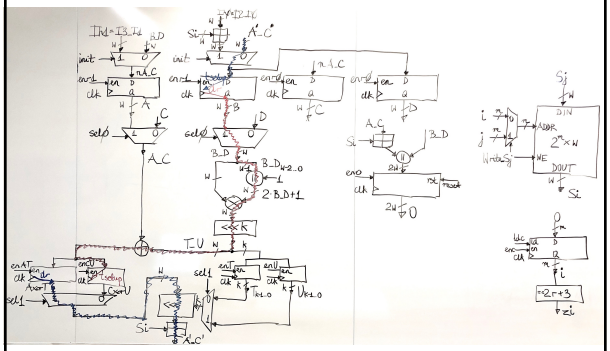
Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[i=0]
D = D + S[i=1]
    for n = 1 to r do
    {
        T = (B*(2B + 1)) <<< k
        U = (D*(2D + 1)) <<< k
        A' = ((A ⊕ T) <<< U) + S[i=2n]
        C' = ((C ⊕ U) <<< T) + S[i=2n + 1]
        (A, B, C, D)  =  (B, C', D, A')
    }
A = A + S[i=2r + 2]
C = C + S[i=2r + 3]
O = (A, B, C, D)
```

62

62

Datapath

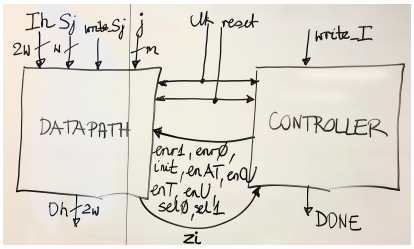


i represents an index of a round key during encryption, not a round number

63

63

Interface with the Division into the Datapath and Controller



64

64

Interface Table

Port	Width	Meaning
clk	1	System clock.
reset	1	System reset – clears internal registers.
Ih	2w	One half of an Input block. I3:I2 first, I1:I0 second.
write_Ih	1	Synchronous write control signal for a half of the input block Ih. After two halves of the input block I are written to the CIPHER unit, the encryption of I starts automatically.
Sj	w	Round key S[j] loaded to one of the two internal memories.
write_Sj	1	Synchronous write control signal for the round key S[j].
j	m	Index of the round key S[j] loaded using input Sj.
Oh	2w	One half of an Output block. O3:O2 first, O1:O0 second.
DONE	1	Asserted for one clock cycle when the output O is ready.

Note:
m is a size of index j. It is a minimum integer, such that $2^m - 1 \geq 2r+3$.

65

65

Pseudocode

i represents an index of a round key during encryption, not a round number

Split input I into four words, I3, I2, I1, I0, of the size of w bits each

```
A = I3; B = I2; C = I1; D=I0
B = B + S[i=0]
D = D + S[i=1]
    for n = 1 to r do
    {
        T = (B*(2B + 1)) <<< k
        U = (D*(2D + 1)) <<< k
        A' = ((A ⊕ T) <<< U) + S[i=2n]
        C' = ((C ⊕ U) <<< T) + S[i=2n + 1]
        (A, B, C, D)  =  (B, C', D, A')
    }
A = A + S[i=2r + 2]
C = C + S[i=2r + 3]
O = (A, B, C, D)
```

66

66

Pseudocode

i represents an index of a round key during encryption, not a round number

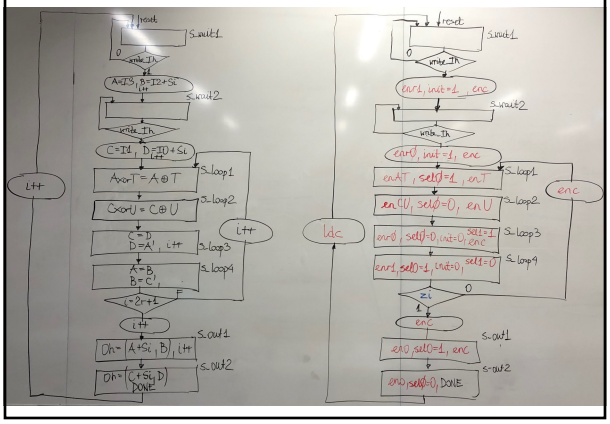
Split input I into four words, I3, I2, I1, I0, of the size of w bits each

A = I3; B = I2; C = I1; D=I0
B = B + S[i=0]
D = D + S[i=1]
for n = 1 to r do
{
T = (B*(2B + 1)) <<< k
U = (D*(2D + 1)) <<< k
A' = ((A ⊕ T) <<< U) + S[i=2n]
C' = ((C ⊕ U) <<< T) + S[i=2n + 1]
(A, B, C, D) = (B, C', D, A')
}
A = A + S[i=2r + 2]
C = C + S[i=2r + 3]
O = (A, B, C, D)

67

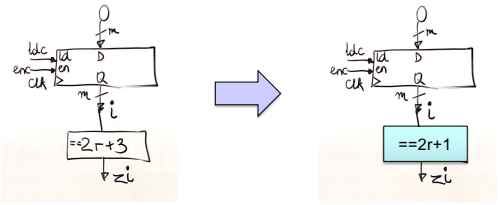
67

ASM Charts



68

Datapath – Loop Counter



69

69

Timing Analysis – Key Setup & Encryption

Notation:
T_{CLK} – clock period in ns
N_M – number of encrypted message blocks I

$T'_{KeySetup}[cycles] = 2r+4$

$T'_{KeySetup}[ns] = (2r+4)\cdot T'_{CLK}$

$T'_{Encryption}(N_M)[cycles] = (4r+4)\cdot N_M$

$T'_{Encryption}(N_M)[ns] = 4\cdot(r+1)\cdot N_M \cdot T'_{CLK}$

70

70

Timing Analysis - Throughput

Notation:
T_{CLK} – clock period in ns
N_M – number of encrypted message blocks I
w – word size

$$Thr'_{Encryption}(N_M)[Gbit/s] = \frac{4 \cdot w \cdot N_M}{4\cdot(r+1)\cdot N_M \cdot T'_{CLK}} = \frac{w}{(r+1)\cdot T'_{CLK}}$$

r & w constants, determined by the security analysis of the cipher
For the security equivalent to AES-128, r=20 & w=32

T'_{CLK} dependent on w, returned by the FPGA tools based on the static timing analysis of the critical path

71

71

Timing Analysis – Throughput Ratio

$$\frac{Thr}{Thr'} = \frac{\frac{4w}{(r+2)\cdot T_{CLK}}}{\frac{w}{(r+1)\cdot T'_{CLK}}} = 4\cdot\frac{r+1}{r+2}\cdot\frac{T'_{CLK}}{T_{CLK}} \approx 4\cdot\frac{1}{2} = 2$$

Throughput decreases by approximately a factor of 2

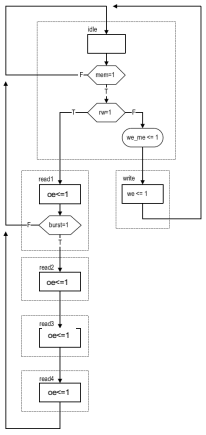
72

72

VHDL Coding Styles

73

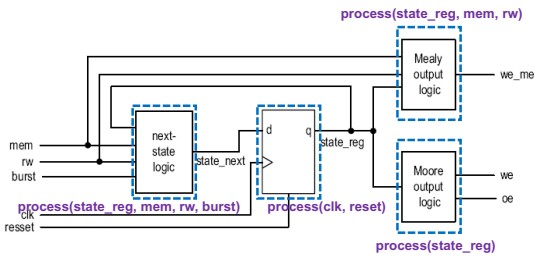
Memory Controller



74

74

Coding Style 1: Acceptable but not recommended



75

75

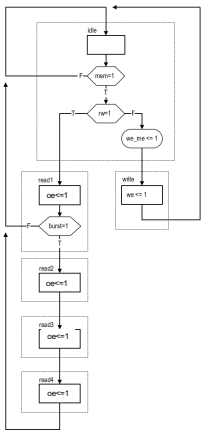
```
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
port (
   clk, reset: in std_logic;
   mem, rw, burst: in std_logic;
   oe, we, we_me: out std_logic);
end mem_ctrl ;

architecture mult_seg_arch of mem_ctrl is
   type mc_state_type is
      (idle, read1, read2, read3, read4, write);
   signal state_reg, state_next: mc_state_type;
begin
   -- state register
   process(clk, reset)
   begin
      if (reset='1') then
         state_reg <= idle;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
```

76

76

Memory Controller



77

77

```
-- next-state logic
process(state_reg, mem, rw, burst)
begin
   case state_reg is
      when idle =>
         if mem='1' then
            if rw='1' then
               state_next <= read1;
            else
               state_next <= write;
            end if;
         else
            state_next <= idle;
         end if;
      when write =>
         state_next <= idle;
```

78

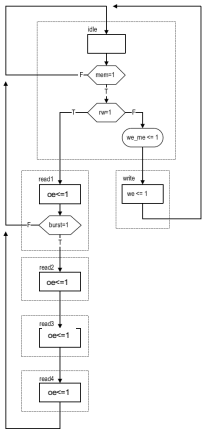
78

```
            when read1 =>
               if (burst='1') then
                  state_next <= read2;
               else
                  state_next <= idle;
               end if;
            when read2 =>
               state_next <= read3;
            when read3 =>
               state_next <= read4;
            when read4 =>
               state_next <= idle;
         end case;
      end process;
```

79

79

Memory Controller



80

80

```
   -- moore output logic
   process(state_reg)
   begin
      we <= '0'; -- default value
      oe <= '0';  -- default value
      case state_reg is
         when idle =>
         when write =>
            we <= '1';
         when read1 =>
            oe <= '1';
         when read2 =>
            oe <= '1';
         when read3 =>
            oe <= '1';
         when read4 =>
            oe <= '1';
      end case;
   end process;
```

81

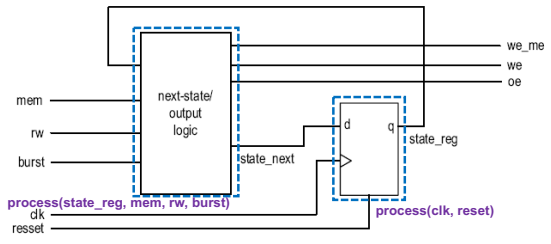
81

```
   -- mealy output logic
   process(state_reg, mem, rw)
   begin
      we_me <= '0'; -- default value
      case state_reg is
         when idle =>
            if (mem='1') and (rw='0') then
              we_me <= '1';
            end if;
         when write =>
         when read1 =>
         when read2 =>
         when read3 =>
         when read4 =>
      end case;
   end process;
end mult_seg_arch;
```

82

82

Coding Style 2 - Recommended



83

83

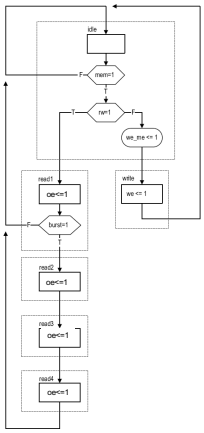
```
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
port (
   clk, reset: in std_logic;
   mem, rw, burst: in std_logic;
   oe, we, we_me: out std_logic);
end mem_ctrl ;

architecture mult_seg_arch of mem_ctrl is
   type mc_state_type is
      (idle, read1, read2, read3, read4, write);
   signal state_reg, state_next: mc_state_type;
begin
   -- state register
   process(clk, reset)
   begin
      if (reset='1') then
         state_reg <= idle;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
```

84

84

Memory Controller



85

85

```
process(state_reg, mem, rw, burst)
begin
  oe <= '0';     -- default values
  we <= '0';
  we_me <= '0';
  case state_reg is
    when idle =>
      if mem='1' then
        if rw='1' then
          state_next <= read1;
        else
          state_next <= write;
          we_me <= '1';
        end if;
      else
        state_next <= idle;
      end if;
    when write =>
      state_next <= idle;
      we <= '1';
```

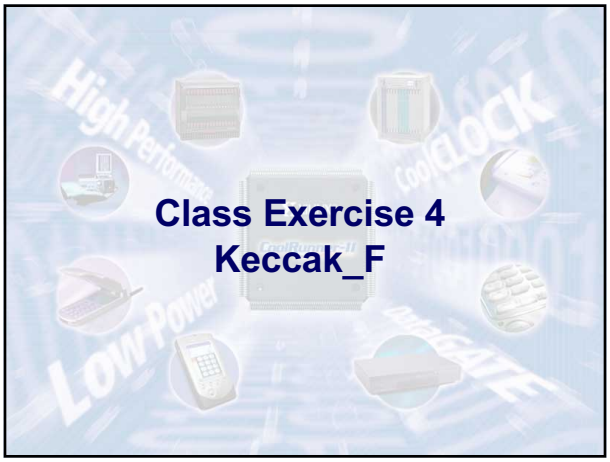
86

86

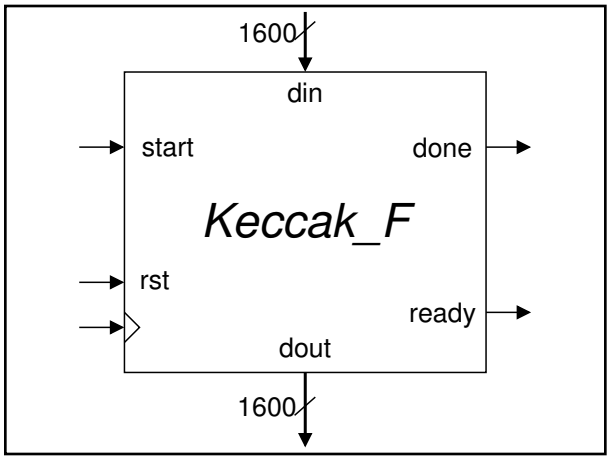
```
    when read1 =>
      if (burst='1') then
        state_next <= read2;
      else
        state_next <= idle;
      end if;
      oe <= '1';
    when read2 =>
      state_next <= read3;
      oe <= '1';
    when read3 =>
      state_next <= read4;
      oe <= '1';
    when read4 =>
      state_next <= idle;
      oe <= '1';
  end case;
end process;
```

87

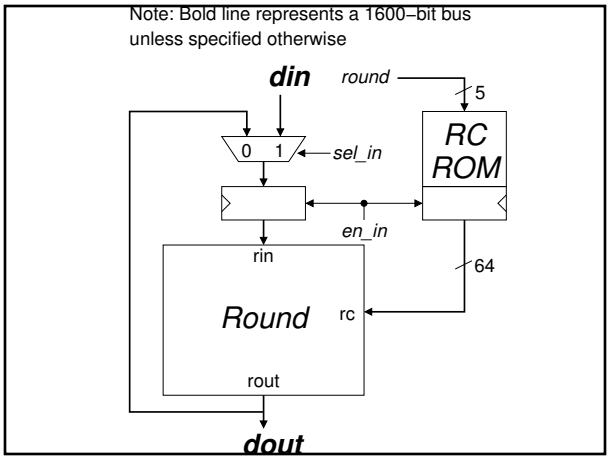
87



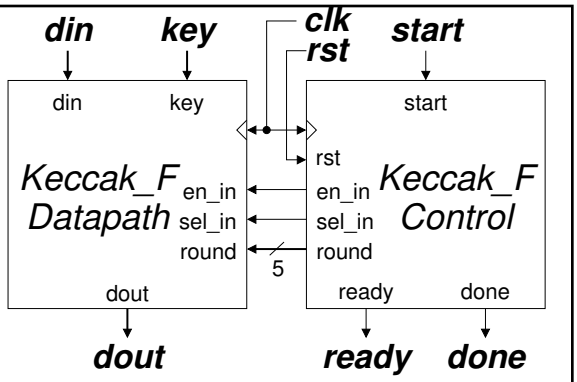
88



89

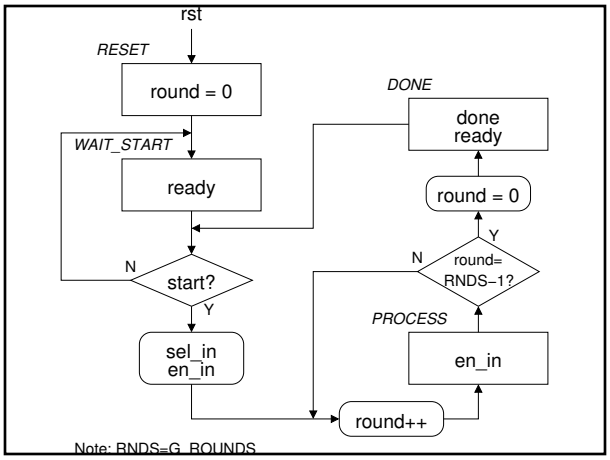


90



Note: Bold line represents a 1600−bit bus unless specified otherwise

91



92

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Keccak_F_Control is
   generic (
      G_RNDS       :   integer := 24
      --! Number of round_rs for the permutation
   );
   port (
      --! Global signals
      clk           : in  std_logic;
      rst           : in  std_logic;
      --! External signals
      start         : in  std_logic;        --! Start signal
      ready         : out std_logic;        --! Ready signal
      done          : out std_logic;        --! Done signal
      --! Internal signals
      round         : out std_logic_vector(4 downto 0);
      sel_in        : out std_logic;
      en_in         : out std_logic
   );
end Keccak_F_Control;
```

93

```
architecture behav of Keccak_F_Control is

   type type_state is (S_RESET, S_WAIT_START, S_PROCESS, S_DONE);
   signal state          : type_state;
   signal state_next     : type_state;
   signal round_r        : std_logic_vector(4 downto 0);
   signal round_next     : std_logic_vector(4 downto 0);

begin
   pctrl:
   process( clk )
   begin
      if rising_edge( clk ) then
         if rst = '1' then
            state      <= S_RESET;
            round_r    <= (others => '0');
         else
            state      <= state_next;
            round_r    <= round_next;
         end if;
      end if;
   end process;
   round <= round_r;
```

94

```
   pcomb:
     process(state, round_r, start)
     begin
        --! Default values
        state_next   <= state;
        round_next   <= round_r;
        ready    <= '0';
        sel_in   <= '0';
        en_in    <= '0';
        done     <= '0';

        case state is
           when S_RESET =>
              round_next  <= (others => '0');
              state_next  <= S_WAIT_START;

           when S_WAIT_START =>
              ready <= '1';
              if (start = '1') then
                 en_in       <= '1';
                 sel_in      <= '1';
                 round_next     <= round_r + 1;
                 state_next     <= S_PROCESS;
              end if;
```

95

```
           when S_PROCESS =>
                en_in <= '1';
                if (round_r = G_RNDS-1) then
                   round_next  <= (others => '0');
                   state_next  <= S_DONE;
                else
                   round_next  <= round_r + 1;
                end if;

           when S_DONE =>
                done    <= '1';
                ready   <= '1';
                if (start = '1') then
                   en_in       <= '1';
                   sel_in      <= '1';
                   round_next  <= round_r + 1;
                   state_next  <= S_PROCESS;
                else
                   state_next  <= S_WAIT_START;
                end if;
        end case;
     end process;
end behav;
```

96